intel®

ARTICLE REPRINT

AR-41

July 1977

An Integral Real-Time Executive For Microcomputers

COMPUTER DESIGN July 1977

9800579A

Single-board microcomputers offer hardware cost-effectiveness for implementing many real-time systems. A compatible, resident, real-time executive program provides savings in software development

An Integral Real-Time Executive For Microcomputers

Kenneth Burgett and Edward F. O'Neil

Intel Corporation
Santa Clara, California

Single-board computers, or microcomputers, that contain central processor, read-write and programmable read-only memory, real-time clock, interrupts, and serial and parallel input/output all on one printed circuit board, have made feasible a whole spectrum of applications which previously could not be economically justified. These microcomputers have also opened up a range of applications where the high functional density of large-scale integration provides advantages over previous solutions such as hardwired logic or relatively expensive minicomputers. While microcomputers readily solve hardware requirements, software for single-board computer applications with real-time characteristics (which are in the majority) has until now been generated individually for each application.

The Intel RMX/80* Real-Time Multi-Tasking Executive simplifies real-time application software development, and at the same time furnishes capabilities optimized for the microcomputer environment. It provides the means to concurrently monitor and control multiple external events that occur asynchronously in real-time. The program framework allows system builders to immediately implement software for their particular applications, and to avoid specific details of system interaction.

Major functions of the executive include system resource access based on task priority, intertask communication, interrupt driven device control, real-time clock control, and interrupt handling. In combination, these functions eliminate the need to implement detailed real-time coordination for specific applications.

Previously, two alternative software approaches were used to solve microcomputer applications. First, many designers created their own operating executive, in- vidually tailored for each application. Obviously, approach was expensive and time-consuming. The sec approach was to use a minicomputer executive which h been adapted to a microcomputer. Since this sof was designed for a different processing environment then "stripped down," it suffered from major in equacies when executed on microcomputers. The alter tive, RMX/80, has been designed specifically to pro a general-purpose real-time executive tailored to Int SBC 80 and System 80 microcomputers.

Real-Time System Requirements

All software design approaches for use in real-time plications include capability for concurrence, prior and synchronization/communication.

Concurrence—Real-time systems monitor and con events which are occurring asynchronously in the ph cal world. Microcomputer software does not know actly when external events will occur; however, it m be prepared to perform the necessary processing u demand, whenever the events actually do occur. Typi ly, interrupts are used to inform the microcomputer an event has occurred. At interrupt time, system con software determines what processing to perform, as as the relative sequence in which processing must place.

*RMX/80™ is a registered trademark of the Intel Corp. S Clara, Calif.

Programs related to external events are processed in an interleaved manner based on interrupt occurrence and priority. For instance, one routine is executing when an interrupt activates, signaling that a higher priority event has occurred. At this point, the routine related to the priority interrupt is started, while execution of the less important routine is discontinued temporarily. When the more important routine is completed, or temporarily halted for some other reason, execution of the less important routine is resumed. In this manner, multiple programs execute concurrently in an interleaved fashion.

Priority—In a real-time environment, certain events require more immediate attention than others because of their significance within the physical world. Immediacy is relative to other processing, and is determined by application requirements. The concept of immediacy or priority, however, is common throughout all real-time microcomputer applications. In priority-based systems, the most important program (one that is not waiting for some physical or logical reason) is the one executing.

A classic illustration of program priority in real-time systems is found in the area of plant control. When the plant begins to fail in a nonrecoverable manner, it is imperative that the plant be shut down as quickly as possible. For this reason, shutdown processing takes priority over all other system demands. Software priority enforces this hardware concept of physical operational events.

Synchronization/Communication—Another common similarity in most real-time systems is the need for synchronization between various events in the physical world which are under microcomputer control. Synchronization is defined as the process whereby one event may cause one or more other events to occur. Communication is the process through which data are sent between input/output (I/O) devices or programs and other programs within the microcomputer system.

An example of the need for synchronization and communication is a microcomputer system for weighing and stamping packages. One part of the system weighs the package, calculates pricing, and releases the package onto a conveyor belt. Price and weight data are communicated to another part of the system which stamps the data onto the package after it arrives at a sensor station. Synchronization is demonstrated by the occurrence of one event—package arrival—causing another event—package stamping—to occur.

Compatible Benefits

To satisfy real-time microcomputer software requirements, the RMX/80 Real-Time Executive software (Fig 1) was designed. This program differs from existing software systems by offering capabilities directly related to the single-board microcomputer environment in which it operates. These capabilities have two major bottom-line benefits compared with equivalent minicomputer systems. First, the executive code is compact enough to allow a large number of real-time applications to be processed on a single microcomputer board. To accomplish this capability, its nucleus is optimized to reside in less than 2k bytes [ie, in a single 16k programmable read-only memory (p/ROM)], thereby allowing up to 10K of onboard memory for application-related software and storage.

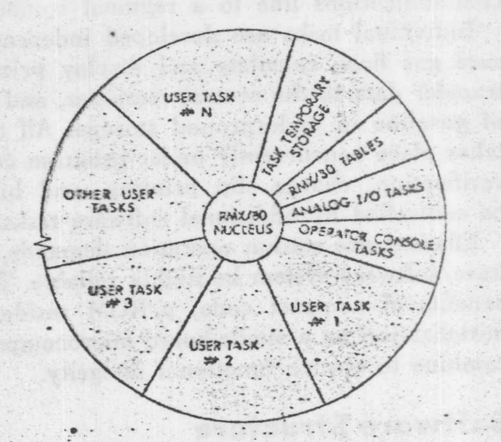

Fig 1 A typical RMX/80 system. Multiple tasks control a given application. Nucleus controls execution of both user and executive tasks through task-to-task communication, real-time clock, priority resolution, and interrupt handling facilities. All tasks within an RMX/80-based application use at least some of these capabilities; other optional executive tasks include debugger, free-space manager, and device control for operator's console, diskette file system, analog subsystems, and high speed mathematics unit

Second, the executive may be p/ROM-resident. When the microcomputer system is powered on, the software system (executive plus application programs) is automatically initialized and begins execution of the highest priority application task. Typical major real-time executives, however, are totally random-access read-write semiconductor memory (RAM)-resident, which means they must be initialized (booted) from a peripheral device, such as diskette, cassette, or communications line, into microcomputer memory. The need for peripheral devices significantly increases the total cost of traditional real-time executive-based solutions.

Sample Application

Functioning as a real-time executive for microcomputers, this software system provides facilities for orderly control and monitoring of asynchronously occurring external events. Although these events may differ widely from application to application, facilities are adaptable to nearly all processes where the microcomputers are used, including process and machine control, test and measurement, data communications, and specialized on-line data processing applications (where one or more terminals access diskette-based data). The executive is particularly useful in dedicated low cost applications which were not economically feasible before the advent of microcomputers. For example, consider the requirement of gas pump control in a service station (Fig 2).

In this station, a microcomputer system operating with RMX/80 concurrently monitors and controls multiple gas pumps, and sends price and volume informa-

tion to one central location. At the same time, information about station operation is being transmitted over a communications line to a regional computer.

Individual tasks are developed independently to measure gas flow, calculate and display price information, transfer data to the central computer, and monitor levels of gasoline in underground storage. All this processing takes place concurrently under program control. (Credit verification, charge slip printing, and billing can also be controlled by additional software tasks.)

Efficient gas station operation demands that the hardware/software system be highly reliable. The compatible benefits of compact code, p/ROM residency, and self-initialization on a single-board microcomputer system all combine to ensure functional integrity.

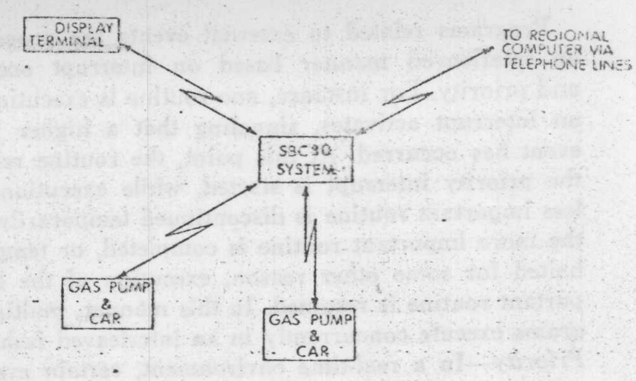

Fig 2 Microcomputer control for gas pump automation. In this example, executive-based system simultaneously controls two pumps, displays information on operator's console, and communicates with regional computer. At a given time, more or fewer functions could be operating concurrently. System expansion can be easily accomplished by adding tasks and modular hardware

Software Structure

RMX/80 simplifies the effort for developing a real-time system, first, by providing many commonly required software functions. Second, its software structure promotes efficient program development. Programmers who are familiar with structured programming will find task orientation both natural and easy to use.

Tasking means that a larger program is divided into a number of smaller, logically independent programs or tasks. The key is to identify functions that may occur concurrently. For example, consider the tasks required for a terminal handler—real-time asynchronous I/O between an operator's CRT terminal and the executive.

Input Handler Task—One task must be ready to accept a data character from the terminal at any time. This is done by responding to an interrupt signal from the terminal and then accepting the data character. The task immediately passes the input character to a subsequent task automatically and then goes back to wait for another interrupt.

Line Buffer Task—As characters are received from the input handler they must be placed into a buffer to form a line. Eventually, the buffer will be filled or the logical end-of-line will be signaled by a carriage return character. At this point, the line buffer must be sent to some other task for processing.

Echo Driver Task—For a full-duplex terminal, it is necessary to return each input character to the terminal for display on the CRT screen. This task waits for a character, which could be sent by either the line buffer or input handler task, and then sends the character to the terminal. It then waits for the next character.

Note that input handler and echo driver are described as waiting for an event. Within the RMX/80, that is literally the case. While they wait, however, system resources are available for other tasks, such as that of the line buffer. Thus, effective processing may occur concurrently with necessary waiting periods. Notice also that a number of other tasks may also be active within the system. In fact, the greater the number of tasks running concurrently, the more effectively system resources are used. Concurrent operation eliminates many time wasting procedures from a real-time system. For example, the executive can eliminate the need for many timing loops where the processor simply executes a no-operation instruction repeatedly while waiting for an event to occur.

Within the executive, tasks not only are logically dependent, they are also physically independent, contending with each other for the use of the processor and other system resources. The executive resolves this contention based on the priority of each task.

In the terminal handler example, it is clear that the input handler must have highest priority, since acceptable performance cannot tolerate the loss of data. Second highest priority is given to the echo driver, so that data appearing on the screen remain coordinated with the input. Lowest priority goes to the line buffer, since that function does not depend directly on an external asynchronous event. There are no particular real-time constraints on the line buffer as long as the input characters are eventually processed.

It is possible to write the entire terminal handler as a single large task instead of as several smaller tasks. However, consideration must be given other high priority tasks operating within the system which may not be able to gain control while a low priority portion of the terminal handler, such as the line buffer task, is executing. Therefore, tasks assigned as high priority are generally kept as short as possible. If the terminal handler were written as one large task, it could tie up the entire processing system for a relatively trivial function.

Task States

Two task states have been implied—running and waiting. A running task is always the task which currently has the highest priority and is not suspended or waiting. A waiting task remains in the wait state until it receives a message or an interrupt for which it is waiting or until a specified time period has passed. The wait period can be timed using the system clock.

A running task may suspend itself on some other task in the system. A suspended task cannot begin execution again until some running task orders it to resume. As an example, a password routine might temporarily suspend the echo driver of the terminal handler so that the password is not displayed. (The password routine must

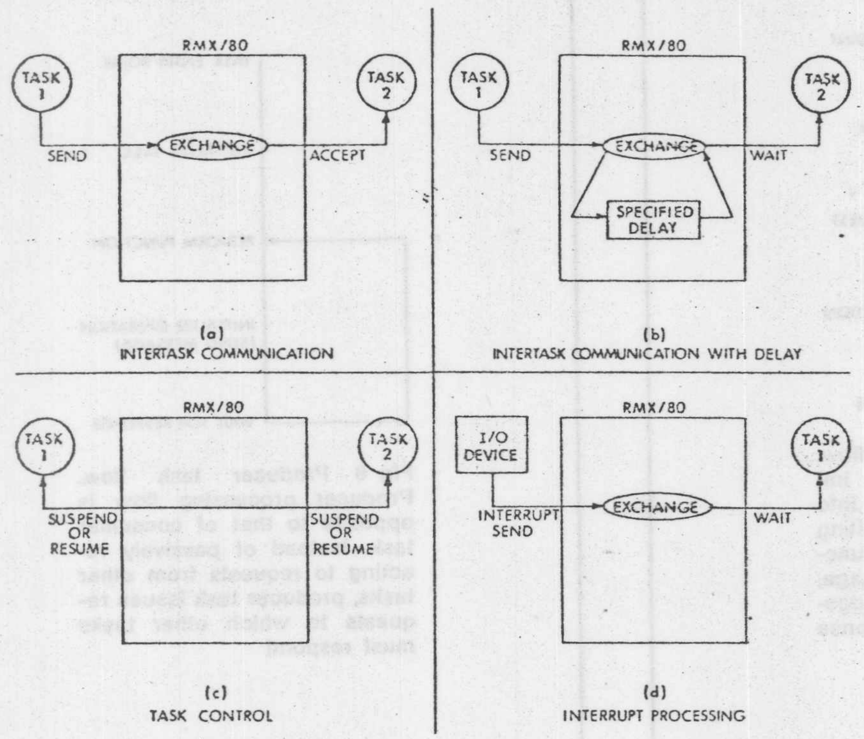

Fig 3 System message exchanges. In intertask communication (a) task 1 sends a message to an exchange, where it is held until task 2 requests message via accept. In intertask communication with delay (b), task 2 waits for a message from task 1 until data are available or until a certain time period has passed, whichever occurs first. In task control (c), any task may suspend or resume any other task. In interrupt processing (d), an I/O interrupt is transformed into a message that task 1 receives via a wait command. Task 1 then performs appropriate interrupt processing

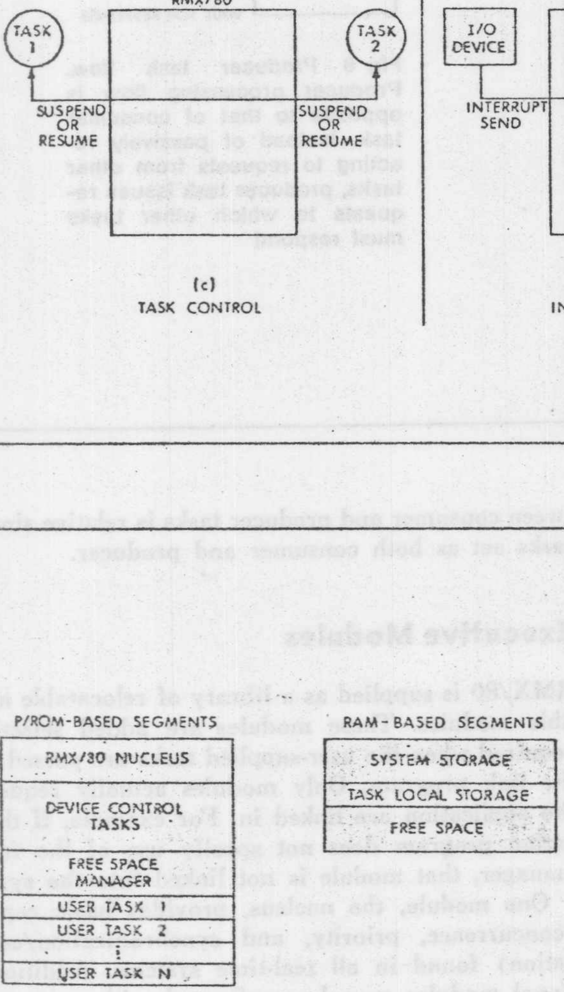

Fig 4 Memory utilization. RMX/80 nucleus, device control task, and free-space allocation modules are linked with user tasks to form a real-time system. Although executive may be RAM-resident, it is designed to reside in p/ROM and uses RAM only for temporary storage and free space. User tasks are provided by user at generation time. RAM may be used by RMX/80 and all associated tasks for temporary storage, including stack.

remove the password from the line buffer, or it will be displayed as soon as execution of the echo driver is resumed.)

A task may also be in the ready state. A ready task is one that would be running except that a task with higher priority temporarily controls the system resources. The executive maintains a list of all tasks that are ready to run. The next task to be run is always the task with the highest priority in the ready list.

The running task relinquishes its control of the system by

(1) Putting itself into a wait state

(2) Suspending itself

(3) Sending a message to a higher priority task, which if it has the highest current priority, becomes the running task

(4) Being preempted by an interrupt to a higher priority task

In the case of an interrupt, the executive saves the status (contents of registers, etc) of the interrupted task so that it will be restarted correctly.

Message Exchanges

Tasks communicate with each other by sending messages (Fig 3). The sending task constructs the message to be sent in RAM or uses a previously assembled message.

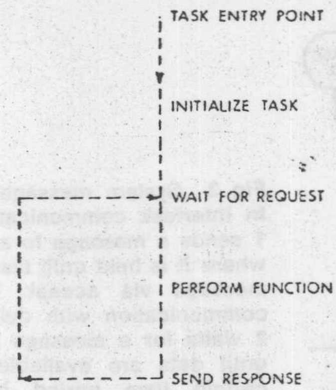

Fig 5 Consumer task flow. Consumer task performs initialization and then drops into cyclic loop, alternately waiting for messages, performing functions requested by message, and sending an acknowledgement in form of a response message

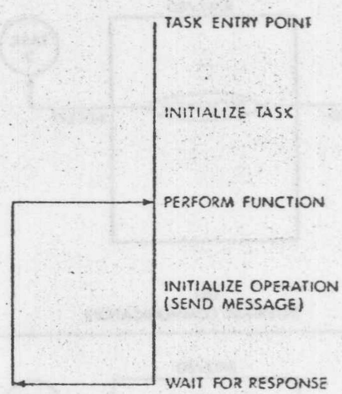

Fig 6 Producer task flow. Producer processing flow is opposite to that of consumer task. Instead of passively reacting to requests from other tasks, producer task issues requests to which other tasks must respond

The sending task then issues a SEND command that posts the address of the message at an exchange.

An exchange is simply a set of lists maintained by the executive. The first list contains the addresses of messages available at that exchange. The second list consists of a list of tasks that are waiting for messages at that exchange. When a task enters a wait state, it specifies the exchange where it expects eventually to find a message. The task may wait indefinitely, or it may specify that it will only wait a specific period of time before resuming execution.

Messages, together with the exchange mechanism, provide for automatic intertask communication and also for task synchronization. For example, a message to a particular task may specify that the task is to send a response to a certain exchange. Thus, the original task may request an acknowledgement response to its message, or it may specify that a message is to be sent to a third task. RMX/80 treats interrupts like messages, the only difference being that interrupts have their own set of exchanges.

Note that the sending and receiving of messages classifies tasks into two types—message consumers and message producers. A consumer task waits for a message, performs an action based on the message, and then returns to the wait state until another message is received. A producer task initiates its function by sending a message to another task, waits for a response, and then sends another message. Figs 5 and 6 graphically illustrate the processing within these two tasks. The distinction between consumer and producer tasks is relative since many tasks act as both consumer and producer.

Executive Modules

RMX/80 is supplied as a library of relocatable and linkable modules. These modules are added selectively as required when the user-supplied tasks are passed through the link program. Only modules actually requested by the application are linked in. For example, if the application program does not specify use of the free-space manager, that module is not linked into the system.

One module, the nucleus, provides basic capabilities (concurrence, priority, and synchronization/communication) found in all real-time systems. Additional, optional modules may be configured with user programs (tasks) to form a complete application software system. These modules include:

Terminal handler—Providing real-time asynchronous I/O between an operator's terminal and tasks running under the RMX/80 executive, the handler offers a line-edit feature similar to that of ISIS-II and an additional type-ahead facility. (ISIS-II is the supervisory system used on the Intellec Development System.)

Free-space manager—This module maintains a pool of free RAM and allocates memory out of the pool upon request from a task. In addition, the manager reclaims memory and returns it to the pool when it is no longer needed.

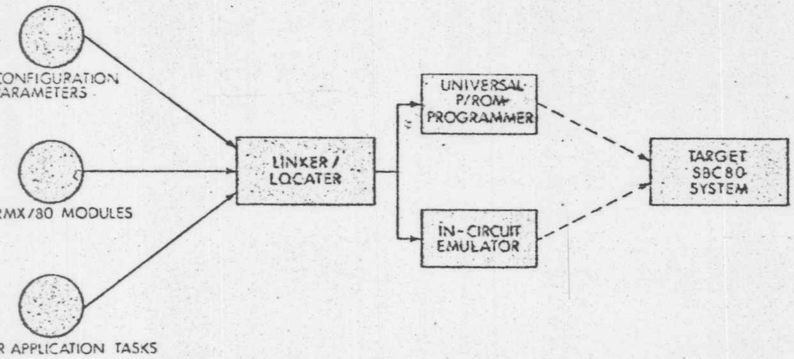

Fig 7 Target microcomputer system. Configuration parameters are linked together with appropriate RMX/80 and user task modules. Resulting program is then transferred to its target SBC 80 system via programmed p/ROMs or is debugged using in-circuit emulation and then transferred

Debugger—Designed specifically for debugging software running under the RMX/80 executive, the debugger is used by linking it to an application program or task. Thus, it can be run directly from the single-board computer's memory. In addition, an in-circuit emulator, such as ICE-80, can be used to load and execute the debugger, providing all resources of the Intellec development system to simplify debugging effort.

Analog interface handlers—Consisting of RMX/80 tasks, these handlers provide real-time control for SBC 711, 724, and 732 systems.

Diskette file systems—Giving RMX/80 users diskette file management capabilities, the diskette driver allows users to load tasks into the system and to create, access, and delete files in a real-time environment without disrupting normal processing. All file formats are compatible with ISIS-II for both single and double density systems.

In addition to application program module or task requirements, the user also supplies a set of generation parameters. These parameters are a set of tables that inform the executive of the number of tasks and exchanges in the system. Fig 7 illustrates the system generation process.

Summary

The significance of RMX/80 to software design parallels the significance of the single-board computer to hardware design. Microcomputers allow designers without extensive experience in digital systems to bring computer processing power into their applications. Similarly, the executive relieves the hardware designer of much software design required for real-time applications. Designed to facilitate growth, since new software needed to support hardware expansions can be supported easily by the addition of new tasks, this executive also substantially reduces recurring costs because it requires a minimum of memory and does not require peripheral bootstrap loading devices. RMX/80 results in economical, shorter, and more flexible software development efforts when designing, building, and verifying real-time user applications.

Bibliography

C. G. Bell, A. Newell, *Computer Structures: Readings and Examples*, McGraw-Hill, New York, 1971

P. Brinch-Hansen, *Operating Systems Principles*, Prentice Hall, 1973

E. W. Dijkstra, "The Structure of the *THE* Multiprogramming Systems," *Communications of the ACM*, May 1968, pp 341-346

E. I. Organick, *The Multics System: An Examination of Its Structure*, MIT Press, Cambridge, Mass, 1972

D. M. Richie, K. Thompson, "The UNIX Time Sharing System," *Communications of the ACM*, July 1974, pp 135-143

Kenneth Burgett, currently software project leader for OEM products and project leader for RMX/80 at Intel Corp, has been involved in system programming for a variety of computers. He holds a BS degree in mechanical engineering from Marquette University.

Edward O'Neil received a BS degree from Louisiana State University and an MBA degree from California State University. Currently software marketing manager for the single-board computer family at Intel Corp, his background includes design and development of real-time operating systems.